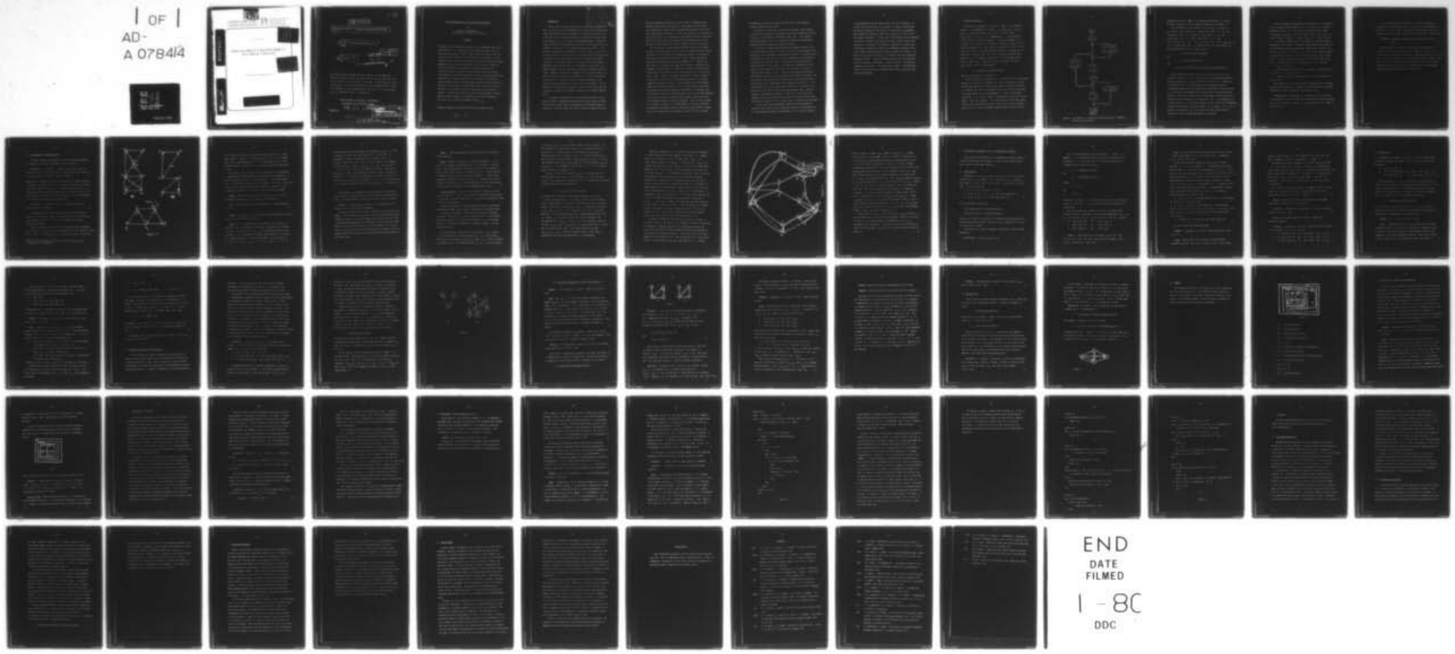
AD-A078 414   MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 5/2
SERIALIZABILITY OF CONCURRENT DATABASE UPDATES,(U)
MAR 79 C H PAPADIMITRIOU   N00014-75-C-0661
UNCLASSIFIED   MIT/LCS/TR-210   NL

1 OF 1
AD-
A 078414



END
DATE
FILMED
1-80
DDC

MIT/LCS/TR-210

SERIALIZABILITY OF CONCURRENT DATABASE UPDATES

Christos H. Papadimitriou

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139 (617) 253-5851

1

14 MIT/LCS/TR-210

6 SERIALIZABILITY OF CONCURRENT DATABASE UPDATES,

10 Christos H./Papadimitriou

12 57

11 Mar 79

March 1979

This report was prepared with the support of the National Science Foundation Grant No. MCS-77-01193 and No. MCS-77-05314. It will appear in the Journal of the ACM. A preliminary version of Sections 2 and 3 was presented at the Conference for Theoretical Computer Science at Waterloo, July 1977, in a joint paper with P.A. Bernstein and J. B. Rothnie.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE

15 N00014-75-C-Ø661
[redacted]
✓NSF-MCS77-Ø1193

409 648 Gm

The Serializability of Concurrent Database Updates*

by

Christos H. Papadimitriou
Massachusetts Institute of Technology

Abstract

A sequence of interleaved user transactions in a database system may not be *serializable*, i.e., equivalent to some sequential execution of the individual transactions. Using a simple transaction model we show that recognizing the transaction histories which are serializable is an NP-complete problem. We therefore introduce several efficiently recognizable subclasses of the class of serializable histories; most of these subclasses correspond to serializability principles existing in the literature and used in practice. We also propose two new principles which subsume all previously known ones. We give necessary and sufficient conditions for a class of histories to be the output of an efficient history scheduler; these conditions imply that there can be no efficient scheduler that outputs all of serializable histories, and also that all subclasses of serializable histories studied above have an efficient scheduler. Finally, we show how our results can be extended to far more general transaction models, to transactions with partly interpreted functions, and to distributed database systems.

*Research supported by NSF Grants MCS-77-01193 and MCS-77-05314.

Page -A-

1. INTRODUCTION

In many situations many users may consult and update a common database. We can think of such independent user transactions as sequences of atomic database operations, interleaved with computations that are local to the user, that is, they do not affect or depend on the current state of the database. It is a function of database management to handle the update and retrieval requests made by the users in such a way so that the resulting overall process is in some appropriate sense correct. It is generally accepted--see, for example, [SLR], [SK], [EGLT], [BPR]--that the right notion of correctness in this context is that of *serializability*. A sequence of atomic user updates/retrievals is called serializable essentially if its overall effect is as though the users took turns, in some order, executing each their entire transaction indivisibly. The simplest example of a non-serializable sequence is a primitive form of a "race". Imagine two users that increment a counter by first sensing its value, and later registering an increased one. If both users retrieve the value of the counter before either of them has updated it, the resulting execution sequence--or *history*--is not serializable. This is because both possible serial executions of these transactions would have resulted in a larger total increment. Naturally, much subtler examples exist.

The appeal of serializability as a correctness criterion is quite easy to justify. Databases are supposed to be faithful models of parts of the world, and user transactions represent instantaneous changes in the world. Since such changes are totally ordered by temporal priority, the only acceptable interleavings of atomic steps of different transactions are those that are equivalent to some sequential execution of these transactions. Another way of viewing serializability is as a tool for ensuring system correctness. If each user transaction is correct--i.e., when run by itself, it is guaranteed to map consistent states of the database to consistent states--and transactions are guaranteed to be intermingled in a serializable way, then the overall system is also correct.

In this paper we consider transactions that consist of two atomic actions: a retrieval of the values of a set of database entities--called the *read-set* of the transaction--followed by an update of the values of another set of entities--the *write-set*. This is exactly the kind of transactions handled by the system SDD-1 [BGRP], [RG]. However, the main reason for considering this model here is that it provides a nice framework for understanding and comparing very different philosophies of serializability that already exist in the literature--e.g., [BS], [SLR], [EGLT], [BGRP]. Despite its apparent simplicity, it yields a theory of serializability that is rich in combinatorial intricacies, and raises interesting complexity questions. Since our model is the most general common restriction of the models in the various references cited above, our negative results apply verbatim to those models. Furthermore, most of our positive results and characterizations are also easily generalizable to more general situations, although their proofs--in many cases their very statements--would be extremely cumbersome. Hence, we view our model as a convenient language, of the right degree of conceptual complexity, for developing and communicating our ideas about serializability, rather than a set of restrictions that enable the proofs of certain theorems.

We formalize our model of transactions in Section 2, where some preliminary results are also proved.

In Section 3 we prove that the question of whether a given sequence of read and write operations corresponding to several transactions (called a *history*) is serializable is NP-complete [AHU], [Ka]. This suggests that, most probably, there is no efficient algorithm that distinguishes between serializable and non-serializable histories.

In Section 4, we study some efficiently recognizable subsets of the set of serializable histories. In other words, we present polynomial-time "heuristics" that approximate the NP-complete predicate of serializability--in a manner quite reminiscent of efficient approximations of NP-complete optimization problems [GJ], [PS]. We show that the two-phase locking strategy [EGLT] and the protocol P3 of [BGRP] are incommensurate special cases of two more general classes called Q and DSR--the latter is related with the model of [SLR]. These two serializability principles are therefore very general (and applicable) new serialization methods. We also introduce the class SSR of histories that can be serialized without reversing the order of temporally non-overlapping transactions; it is not known whether this class is efficiently recognizable. In Section 5, we observe that the quite intricate interrelations among these interesting classes are simplified considerably if some "static" restrictions are imposed on the read- and write-sets. We point out there that the simple serializability theory of [SLR] is due to such a restriction of their model.

For all efficiently recognizable classes of histories studied in Sections 4 and 5 there is also an efficient *scheduler*; an algorithm, that is, which takes any history and transforms it to its closest (according to some appropriate metric) history within the class considered. In Section 6 we show that this is no accident: a class of histories has an efficient scheduler if and only if it is efficiently recognizable, plus a regularity condition, namely that its set of *prefixes* is also efficiently recognizable. By this result, the complexity theory developed in Sections 3 through 5 is practically relevant, because the practical question of the existence of an efficient scheduler for a given class of histories is explicitly linked to the complexity properties of the class. Another implication is the negative result that, unless $P = NP$, there is no efficient "serializer" of histories, and hence considering efficient but more restrictive schedulers--such as the ones discussed above--is a reasonable alternative. Finally, Section 7 concludes our treatment of the subject. We discuss there a number of possible extensions of our results such as to general (multi-step) transactions and distributed databases.

2. DEFINITIONS-NOTATION

A *history* is a quadruple $h = (n, \pi, V, S)$, where n is a positive integer; π is a permutation of the set $\Sigma_n = \{R_1, W_1, R_2, W_2, \ldots, R_n, W_n\}$--that is, a one-to-one function $\pi : \Sigma_n \rightarrow \{1, 2, \ldots, 2n\}$--such that $\pi(R_i) < \pi(W_i)$ for $i = 1, 2, \ldots, n$ (a permutation π is represented by $<\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(2n)>$); finally, S is a function mapping Σ_n to 2^V, where V is a finite set of *variables*. Each pair (R_i, W_i) will be called a *transaction* T_i. $S(R_i)$ will be called the *read set* of T_i, and $S(W_i)$ its *write set*. We shall represent histories in a compact way by exhibiting π, with the sets $S(\cdot)$ given in brackets following each element of Σ_n. For example, the history $h = (3, <R_1, R_2, W_1, R_3, W_2, W_3>, \{x,y\}, S)$ where $S(R_1) = S(R_3) = \{x\}$, $S(R_2) = \emptyset$, $S(W_3) = \{y\}$, and $S(W_1) = S(W_2) = \{x,y\}$ is represented as

$$h = R_1[x]R_2W_1[x,y]R_3[x]W_2[x,y]W_3[y].$$

The set of all histories is denoted by H.

We can think of each transaction T_i as starting with an instantaneous reading of the values in the variables in $S(R_i)$, performing a possibly lengthy local computation and then instantaneously recording the results in a different set $S(W_i)$ of variables. We do not look into the details of the exact nature of the local computation. In fact, we view each transaction T_i as a set of $|S(W_i)|$ uninterpreted $|S(R_i)|$-ary function symbols $\{f_{ij} : j = 1, \ldots, |S(W_i)|\}$. π is the sequence in which these atomic read and write operations take place. Thus, a history can be viewed as a special case of a fork-join parallel program schema, in which the local

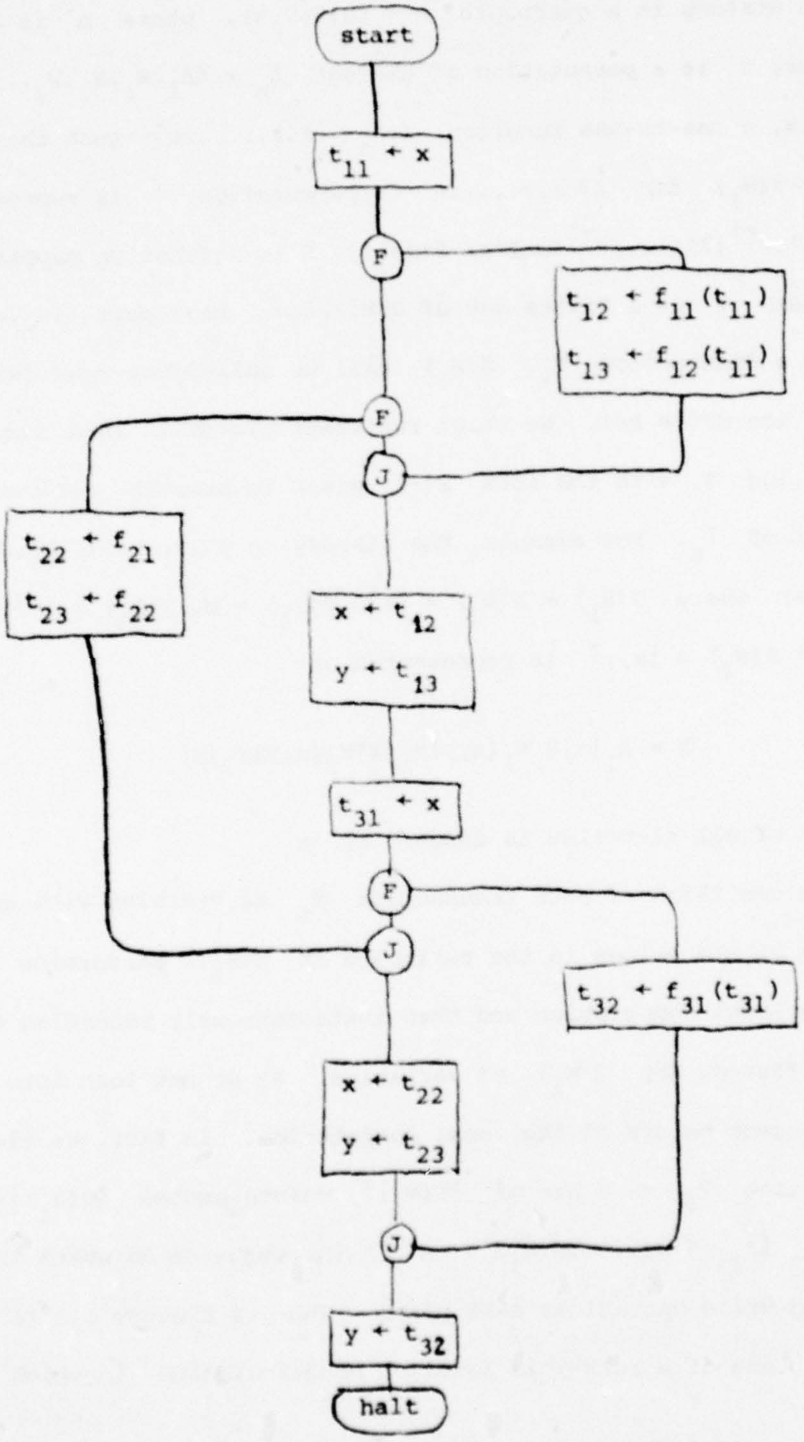


Figure 1. The history $h = R_1[x]R_2W_1[x,y]R_3[x]W_2[x,y]W_3[y]$ viewed as a parallel program schema.

computations involve a number of local temporary variables t_{ij} and are executed in parallel with other read-write operations (see Figure 1).

The *concatenation* of two histories $h_1 = (n,\pi,V,S)$, $h_2 = (m,\rho,V,T)$ is a history $h_1 \circ h_2 = (n+m,\tau,V,P)$, where $P(W_i) = S(W_i)$ if $i \leq n$, and $P(W_i)=T(W_{i-n})$ for $i > n$. Similarly, $P(R_i) = S(R_i)$ if $i \leq n$, and $P(R_i) = T(R_{i-n})$ for $i > n$. Also $\tau(W_i) = \pi(W_i)$ if $i \leq n$, and $\tau(W_i) = \rho(W_{i-n})+2n$ for $i > n$, $\tau(R_i) = \pi(R_i)$ for $i \leq n$, $\tau(R_i) = \rho(R_{i-n})+2n$ for $i > n$. In other words $h_1 \bullet h_2$ is a juxtaposition of the two histories, only with the transactions of h_2 renamed. Thus, if

$$h_1 = R_1[x]R_2[y]W_2[y]R_3W_1[z]W_3[y]$$

and

$$h_2 = R_1[x,y]R_2[x]W_1[y]W_2[z] \quad ,$$

then

$$h_1 \circ h_2 = R_1[x]R_2[y]W_2[y]R_3W_1[z]W_3[y]R_4[x,y]R_5[x]W_4[y]W_5[z].$$

We say that two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are *equivalent* (written $h_1 \equiv h_2$) if and only if the corresponding schemata are (strongly) equivalent. In other words, given any set of $|V|$ domains for the variables, any set of initial values for the variables from the corresponding domains, and, furthermore, any interpretation of the functions f_{ij}, the values of the variables are identical after the execution of both histories. Notice that our definition of equivalence requires that the two histories involve the same set of transactions. Thus $h_1 = R_1[y]R_2W_2[x]W_1[x]$ is not equivalent to $h_2 = R_1[y]W_1[x]$, despite the fact that their corresponding schemata are equivalent (essentially because T_2 is "dead" in h_1). This is a matter of convenience, and little change to our derivations would be necessary in order to broaden equivalence in this sense.

To give a syntactic characterization of equivalence, it is necessary to first introduce some terminology. Let $h = (n,\pi,V,S)$ be a history. The *augmented version* of h is the history $\bar{h} = (n+2,\bar{\pi},V,\bar{S})$, where $\bar{\pi} = \langle R_{n+1}, W_{n+1}, \pi, R_{n+2}, W_{n+2} \rangle$ and $\bar{S}(R_i) = S(R_i)$, $\bar{S}(W_i) = S(W_i)$ for $i \leq n$, and also $\bar{S}(R_{n+1}) = \bar{S}(W_{n+2}) = \emptyset$, $\bar{S}(W_{n+1}) = \bar{S}(R_{n+2}) = V$. In other words, $\bar{h}$ is h preceded by a transaction that initializes all variables without sensing any, and followed by a transaction that reads the final values of all the variables, without changing them. Suppose that $x \in S(R_i)$. We say that R_i *reads* x *from* W_j *in* h if W_j is the latest occurrence of a write symbol before R_i in $\bar{h}$ such that $x \in S(W_j)$. Notice that since $\bar{h}$ contains W_{n+1} with $S(W_{n+1}) = V$, such a write symbol always exists. The definition of a *live* transaction in h is as follows:

a. T_{n+2} is live in h.

b. If for some live transaction T_j, R_j reads a variable from W_i in h, then T_i is also live in h.

c. The only kinds of live transactions in h are defined by (a) and (b) above.

The following is now a simple syntactic characterization of history equivalence, essentially a restatement of the characterization of schema equivalence in terms of Herbrand interpretations, [LPP]:

PROPOSITION 1. Two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are equivalent if and only if they have the same sets of live transactions, and a live R_i reads x from W_j in h_1 if and only if R_i reads x from W_j in h_2. □

One of the implications of Proposition 1 is that equivalence of histories can be decided efficiently. The sets of live transactions can be found in $O(n\cdot|V|)$ time by applying the recursive definition given above, and so can the *reads from* relation for transactions. Hence we have:

COROLLARY. Equivalence of histories can be decided in $O(n\cdot|V|)$ time. □

The main theme of this paper is the notion of *serializability*. A history $h=(n,\pi,V,S)$ is *serial* if $\pi(W_i)=\pi(R_i)+1$ for all $i=1,2,\ldots,n$; in other words, a history is serial if R_i immediately preceeds W_i in it for $i=1,\ldots,n$. A history h is *serializable* (notation: $h\in SR$) if and only if there is a serial history h_s such that $h\equiv h_s$. In the next section we shall present a syntactic characterization of serializable histories analogous to (and based on) Proposition 1.

3. THE COMPLEXITY OF SERIALIZABILITY

In order to examine the complexity of the serializability problem, we need first to introduce some graph-theoretic terminology.

DEFINITION 1. A *polygraph** $P = (N,A,B)$ is a digraph (N,A) together with a set B of *bipaths*; that is, pairs of arcs--not necessarily in A--of the form $((v,u),(u,w))$ such that $(w,v) \in A$. □

Alternatively, a polygraph (N,A,B) can be viewed as a family $\mathcal{D}(N,A,B)$ of digraphs. A digraph (N,A') is in $\mathcal{D}(N,A,B)$ if and only if $A \subseteq A'$, and for each bipath $(a_1,a_2) \in B$, A' contains at least one of a_1, a_2. Polygraphs will be represented schematically as in Figure 2a. Arcs in A will be drawn as ordinary arrows, and pairs of arcs in B will be marked by a circular arc centered on their common node.

DEFINITION 2. A polygraph (N,A,B) is *acyclic* if there is an acyclic digraph in $\mathcal{D}(N,A,B)$. □

For example, the digraph of Figure 2b is both in $\mathcal{D}(N,A,B)$ and acyclic; it follows that (N,A,B) of Figure 2a is acyclic. Notice that for a polygraph (N,A,B) to be acyclic, the digraph (N,A) must definitely be acyclic.

Given any history $h = (n,\pi,V,S)$ we are going to define a polygraph $P(h) = (N,A,B)$. N is the set of live transactions of $\bar{h}$, the augmented version of h. First, A contains the arcs $\{(T_{n+1},v): v \in N-\{T_{n+1}\}\}$, and also the

*We insist on this terminology only because it has already become notorious for its impropriety.

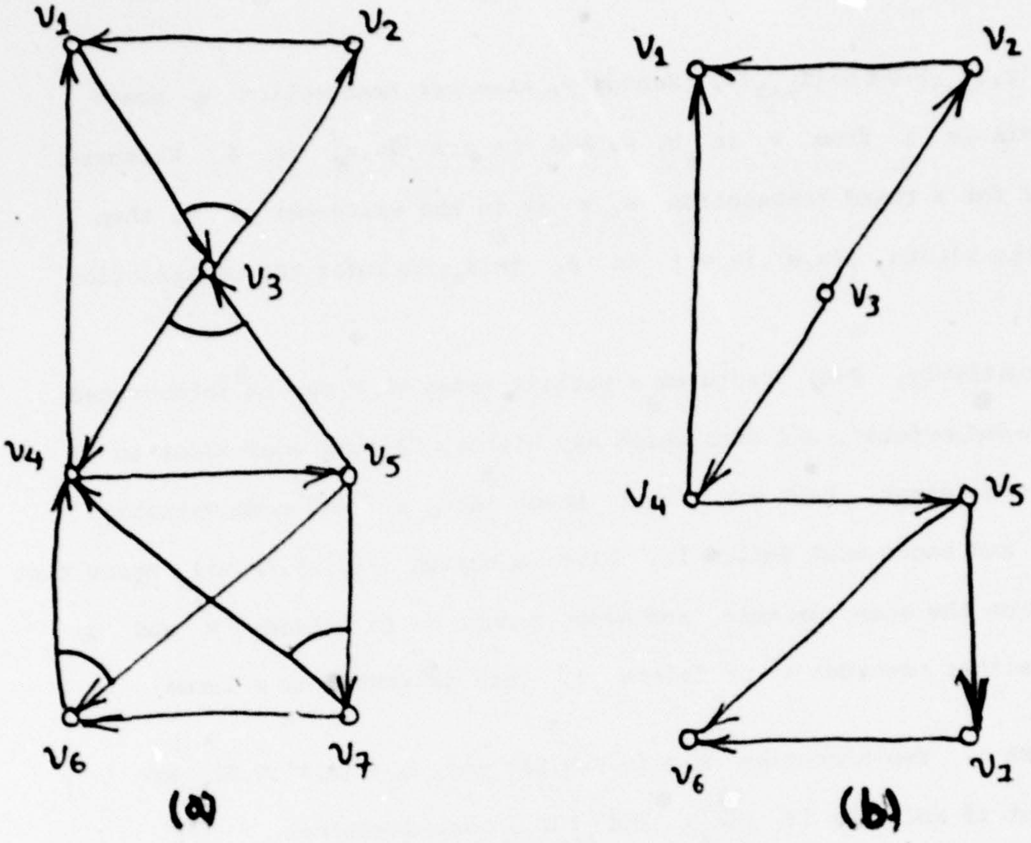


Figure 2

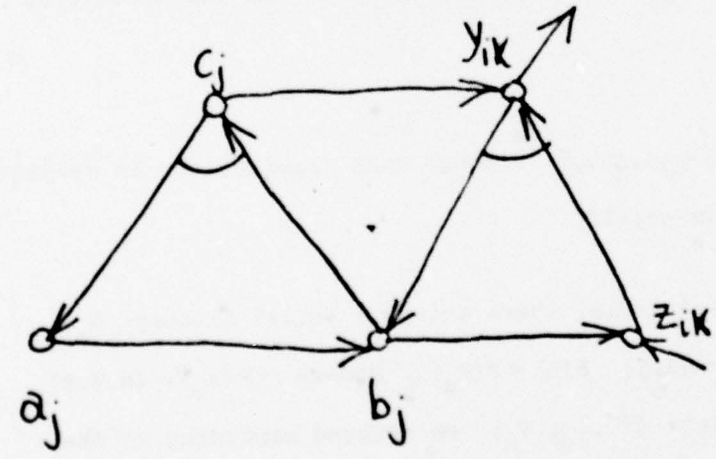


Figure 3

arcs $\{(v,T_{n+2}):v \in N-\{T_{n+2}\}\}$. Secondly, whenever transaction u reads some variable x from v in h, we add the arc (v,u) in A. Furthermore, if for a third transaction w, x is in the write-set of w, then we add the bipath $((u,w),(w,v))$ in B. This concludes the construction of $P(h)$.

Intuitively, $P(h)$ captures a partial order that can be interpreted as "happened before", and with which any history that is equivalent to h must be consistent. Each arc (v,u) means that u read some variable from v and hence must follow it. Also, a bipath $((u,w),(w,v))$ means that w writes on the same variable, and hence cannot be in between v and u; it must either precede v or follow u. This is stated as a lemma:

LEMMA 1. Two histories $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ are equivalent if and only if $P(h_1)$ and $P(h_2)$ are identical.

Proof. Both directions follow from Proposition 1 and the definition of $P(h)$. □

LEMMA 2. A history $h = (n,\pi,V,S)$ without dead transactions is serializable if and only if $P(h)$ is acyclic.

Proof. If h is serializable, there exists a serial history h_s such that $h \equiv h_s$ or, by Lemma 1, $P(h) = P(h_s)$. However $P(h_s) = (N,A,B)$ is acyclic. To see this, let $(T_1,\ldots,T_n)$ be ordered according to their occurrence in h_s. We construct a digraph $(N,A') \in \mathcal{D}(P(h_s))$ as follows: A' contains the arcs in A, and for each bipath $((T_i,T_j),(T_j,T_k))$ in B we add to A the arc (T_i,T_j) if $i<j$, or (T_j,T_k) if $j<k$. To

show that exactly one of these must occur, recall that in h_s T_i reads a variable $x \in S(W_j)$ from T_k, and hence $k < i$, and not $k < j < i$. Consequently the above construction yields a digraph (N,A') in $\mathcal{D}(P,A,B)$. Next, notice that (N,A') is acyclic since it is a subgraph of the total order $(T_{n+1},T_1,\ldots,T_n,T_{n+2})$. So, $P(h)$ is also acyclic.

Now, let (N,A') be an acyclic digraph in $\mathcal{D}(P(h))$. The serial history h_s resulting from topologically sorting (N,A') is then equivalent to h. This follows from Proposition 1 and from the fact that since one of the two arcs of each bipath in B is in A', all transactions in h_s read all variables from the same transaction in h as they do in h_s. □

Unfortunately, the combinatorial characterization of serial reproducibility shown in Lemma 2 does not directly suggest an efficient test. In fact, the theorem below is strong evidence that no such test exists.

THEOREM 1. Testing whether a history h is serializable is NP-complete, even if h has no dead transactions.

In order to proceed with the proof of Theorem 1 we first need another lemma. It is well known (see [AHU],[Ka]) that the satisfiability problem of Boolean formulas in conjunctive normal form with two or three literals in each clause (abbreviated SAT) is NP-complete. We can show that a more restricted version of this problem is still NP-complete. Call a clause *mixed* if it contains both variables and negations of variables, and call a formula *noncircular* if at most one of the occurrences of each variable is in a mixed clause.

LEMMA 3. SAT is NP-complete even if the formulae are restricted to be noncircular.

Proof. Consider any instance F of SAT and a variable x in it. Let m be the number of occurrences of x in the formula F, and let $x_1, x_2, \ldots, x_m$ be new variables. We replace x in its first occurrence by x_1, in its second by $\bar{x}_2$, in its third by x_3, *etc.* Finally, we add the clauses $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge \ldots$, which is the conjunctive normal form of $x_1 \equiv \bar{x}_2 \equiv x_3 \equiv \bar{x}_4 \equiv \cdots$. Repeating this for all variables, we observe that the resulting formula is trivially noncircular, and the construction requires only a polynomial amount of time. □

Proof of Theorem 1. The set of SR histories is definitely in *NP*, since to show that h is SR, one only needs to construct a serial history h_S (of length not greater than that of h), and check by Proposition 1 that h and h_S are equivalent.

We will next show that a known NP-complete problem, the noncircular SAT problem of Lemma 3 above, reduces to *SR-testing in polynomial time.*

Given any such formula F, we are going to construct a polygraph $P_F = (N,A,B)$ such that P_F is acyclic if and only if F is satisfiable. We will then show that P_F can be considered as $P(h)$ for a suitable history h, without dead transactions. In view of Lemma 2, this will conclude the proof.

We start from the construction of $P_F = (N,A,B)$. F has m clauses $C_1, \ldots, C_m$ and involves n Boolean variables $x_1, \ldots, x_n$. Each clause C_i consists of three literals $\lambda_{i1} \vee \lambda_{i2} \vee \lambda_{i3}$, where λ_{jk} is either a variable or a negation of one. N contains the nodes a_j, b_j, c_j for each variable

x_j, and y_{ik}, z_{ik}, $k = 1,..,m_i$ for each clause C_i with m_i literals. For each variable x_j we add the arc (a_j,b_j) to A, and the bipath $((b_j,c_j),(c_j,a_j))$ to B. For each clause C_i, we add the arcs $(y_{ik},z_{i,k+1})$ (addition mod m_i) to A. Finally, if $\lambda_{ik} = x_j$, we add the arcs (c_j,y_{ik}) and (b_j,z_{ik}) to A, and the bipath $((z_{ik},y_{ik}),(y_{ik},b_j))$ to B. If $\lambda_{ik} = \bar{x}_j$, then we add the arcs (z_{ik},c_j) and (y_{ik},a_j) to A, and the bipath $((a_j,z_{ik}),(z_{ik},y_{ik}))$ to B. For example, if the literal λ_{ik} is x_j, the subpolygraph of Figure 3 will appear in P_F.

Finally, we add to N the nodes n_0,n_c and n_f, together with the arcs $(n_0,n),(n,n_c)$ and (n,n_f) for all $n \in N-\{n_0,n_c,n_f\}$, and also the arc (n_c,n_f). This concludes the construction of P_F. In Figure 4a we illustrate the construction for the Boolean formula

$$F = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3).$$

For simplicity, in Figure 4 we have omitted the nodes n_0 and n_f.

We will now argue that P_F is acyclic if and only if F is satisfiable. Suppose that P_F is acyclic. This means that there is an acyclic digraph $(N,A') \in \mathcal{D}(P_F)$. Obviously, for each j, exactly one of the edges (b_j,c_j) and (c_j,a_j) is in A'. Think the fact that $(c_j,a_j) \in A'$ means that x_j is assigned the value *true*. We may immediately note that if a literal λ_{ik} is given the value *false* by this assignment, the corresponding arc (z_{ik},y_{ik}) is also in A', since otherwise, a cycle of the form (c_j,y_{ik},b_j)--or (z_{ik},c_j,a_j) if $\lambda_{ik} = \bar{x}_j$--would exist in (N,A'). Hence, the only way for (N,A') not to have a cycle of the form $(z_{i1},y_{i1},z_{i2},\ldots,y_{i3})$ is that at least one literal in each clause is assigned the value *true*, which means that F is satisfiable.

Conversely, suppose that F is satisfied by some truth assignment T. We will construct an acyclic digraph $(N,A') \in \mathcal{D}(P_F)$. A' contains all of A and the arcs (c_j,a_j) if $T(x_j)=$ *true*, (b_j,c_j) if $T(x_j)=$ *false*, and the arcs (z_{ik},y_{ik}) if $T(\lambda_{ik})=$ *false*, (y_{ik},b_j) if $\lambda_{ik}=\bar{x}_j$ and $T(x_j)=$ *true*, and (a_j,z_{ik}) if $\lambda_{jk}=x_j$ and $T(x_j)=$ *false*. Obviously, (N,A') is in $\mathcal{D}(P_F)$; the claim is that it is acyclic. We first note that since F is by hypothesis noncircular, (N,A) is acyclic. This is because by the construction of A, the clauses containing variables only or negations only correspond to node sets with only incoming or, respectively, only outgoing arcs; node sets corresponding to mixed clauses have both incoming and outgoing arcs, but no two such node sets are reachable from each other in (N,A), by F's noncircularity; it follows that (N,A) is indeed acyclic. It is easy to check that the arcs in $A'-A$ can harm the digraph's acyclicity only by introducing a $(z_{i1},y_{i1},\ldots,y_{i3})$ cycle; however, this would mean that some clause has no *true* (under T) literal, and hence T does not satisfy F, a contradiction. In Figure 4 we show in broken lines the arcs of an acyclic digraph in $\mathcal{D}(P_F)$; this digraph corresponds to the truth assignment $T(x_1)=$ *true*, $T(x_2)=$ *false*, $T(x_3)=$ *false* which satisfies F.

In order to conclude the proof we need to construct a history h such that $P(h)=P_F$. All nodes of P_F correspond to distinct transactions. To construct the read and write sets of the transactions (except for n_0, n_c and n_f), we start by having all read sets empty, and a variable x_v in the write set of each transaction v. For each arc $(v,u) \in A$ we add a variable x_{vu} to the write set of v and the read set of u, and for each bipath $((v,u),(u,w)) \in B$ we add x_{wv} to the write set of u. Finally,

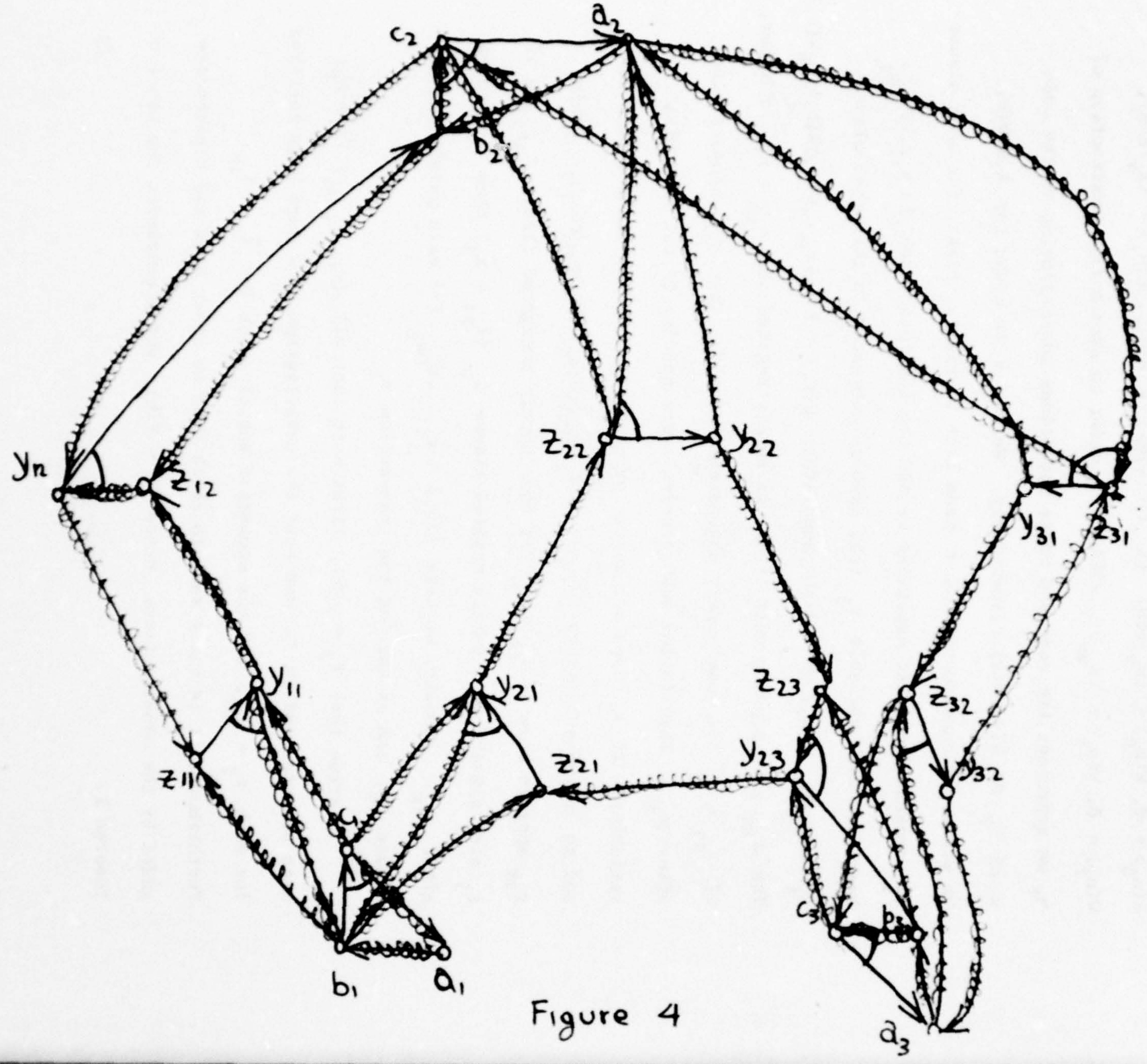

c2
a2
b2
z22
y22
y12
z12
y31
z31
y11
y21
z21
z23
y23
z32
y32
z11
c1
c3
b3
b1
a1
a3

Figure 4

$R(n_0) = \emptyset$, $W(n_0) = \{x_v : v \in N\}$ $\{x_{uv} : (u,v) \in A\} = R(n_f)$, $R(n_c) = \{x_v : u \in N\}$, $W(n_f) = \emptyset$, $W(n_c) = \{x_{uv} : (u,v) \in A\}$. In order to sketch the construction of h, we represent the read and write operations corresponding to the node v of P_F by R(v),W(v) respectively. We use v to stand for R(v)W(v). We start the construction of h from left to right. First, for each clause C_i consisting of just negations we add the subhistory $h(C_i) = y_{i1} \ldots y_{im_i}$. Next, for each variable x_j that appears unnegated in the mixed clause C_ℓ $(1.\ell_j)_{\ell R} = x_j)$ we add the subhistory $h(x_j) = R(a_j) z_{im} c_j W(a_j) R(b_j) y_{\ell R} W(b_j)$. The z_{im} part appears only if C_i is purely negated and $\lambda_{im} = \bar{x}_j$. Further, if $\lambda_{pq} = x_j$ for some purely unnegated clause C_p then y_{pq} appears also after $y_{\ell k}$. Then follow subhistories corresponding to the remaining variables. If x_j does not appear unnegated in a mixed clause, then we add to h the subhistory $h(x_j) = R(a_j) z_{im} c_j W(a_j) R(b_j) y_{\ell R} W(b_j)$. Again, $y_{\ell R}$ appears only if $\lambda_{\ell R} = x_j$ for some purely unnegated clause C_ℓ, and if x_j also appears in a purely negated clause C_p $(\lambda_{pq} = \bar{x}_j)$ then z_{pq} comes after z_{im}. Finally, we have $h(C_i) = z_{i1} \ldots z_{im_i}$ for each purely negated clause C_i, and at the end the transaction n_c.

To argue that $P_F = p(h)$, first note that all (y_{ij}, z_{ij+1}) (mod m_i) arcs are realized by h, and that the subpolygraph of Figure 3 is realized for each $x_j = \lambda_{ik}$, and the symmetric subpolygraph for $\bar{x}_j = \lambda_{iR}$. Furthermore, it is quite easy to check that no other arcs and bipaths are added by the construction. Hence $P_F = P(h)$, which completes the proof of Theorem 1. □

4. EFFICIENTLY RECOGNIZABLE CLASSES OF SERIALIZABLE HISTORIES

Given that SR is NP-complete, it is reasonable to look for subsets of SR that are efficiently recognizable. In this section we study several such classes of serializable histories.

4.1 The Class DSR

DEFINITION 3. Let $h_1 = (n,\pi,V,S)$ and $h_2 = (n,\pi',V,S)$ be histories. We write that $h_1 \sim h_2$ whenever $\pi(\sigma) = \pi'(\sigma)$ for all $\sigma \in \Sigma_n$ except for two elements $\sigma_1, \sigma_2 \in \Sigma_n$ with $\pi(\sigma_1) = \pi'(\sigma_2) = j$, $\pi(\sigma_2) = \pi'(\sigma_1) = j+1$ for some $1 \le j \le n-1$, and either

a. $\sigma_1 = R_i$, $\sigma_2 = R_j$ for some $i, j \le n$, or

b. $\sigma_1 = R_i$, $\sigma_2 = W_j$, $i \neq j$, $i, j \le n$, and $S(R_i) \cap S(W_j) = \emptyset$, or

c. $\sigma_1 = W_i$, $\sigma_2 = W_j$, $i, j \le n$, and $S(W_i) \cap S(W_j) = \emptyset$. □

As an illustration, we have that

$$R_1[x]R_2[x]W_1[x]W_2[y] \sim R_1[x]R_2[x]W_2[y]W_1[x] \sim$$

$$R_2[x]R_1[x]W_2[y]W_1[y] \sim R_2[x]W_2[y]R_1[x]W_1[x] \quad ,$$

because at each step the next history is obtained from the previous one by switching two adjacent symbols obeying one of the conditions (a), (b) and (c) of Definition 3 above.

The following is a direct consequence of Proposition 1 and the above definition:

PROPOSITION 3. If $h_1 \sim h_2$, then $h_1 \equiv h_2$. □

Let $\overset{*}{\sim}$ be the reflexive-transitive closure of $\sim$. Since $\sim$ is symmetric, $\overset{*}{\sim}$ is an equivalence relation which is, by Proposition 3, a restriction of $\equiv$. We can show that $\overset{*}{\sim}$ is a proper restriction of $\equiv$ by observing that for the two histories

$$h_1 = R_3[x]R_1W_1[x]R_2[y]W_2W_3[y]$$

and

$$h_2 = R_2[y]R_3[x]W_2\,W_3[y]R_1W_1[x]$$

we have

$$h_1 \equiv h_2 ,$$

but

$$h_1 \overset{*}{\not\sim} h_2 .$$

We say that the history h is *D-serializable (DSR)* if there is a serial history h_S such that $h \overset{*}{\sim} h_S$. Obviously, if a history is DSR, it is certainly SR.

We can associate with a history $h=(n,\pi,V,S)$ a digraph $D(h)$ defined as follows: The nodes of $D(h)$ are the transactions $\{T_1,\ldots,T_n\}$ of h, and the pair (T_i,T_j) is an arc of $D(h)$ if and only if either

a. $S(R_i)\cap S(W_j) \neq \emptyset$ and $\pi(R_i) < \pi(W_j)$, or

b. $S(W_i)\cap S(R_j) \neq \emptyset$ and $\pi(W_i) < \pi(R_j)$, or

c. $S(W_i)\cap S(W_j) \neq \emptyset$ and $\pi(W_i) < \pi(W_j)$.

LEMMA 4. Suppose that for two histories $h_1=(n,\pi,V,S)$ and $h_2=(n,\pi',V,S)$ $D(h_1)$ and $D(h_2)$ have no cycles of length 2. Then $h_1 \overset{*}{\sim} h_2$ if and only if $D(h_1)=D(h_2)$.

Proof. It should be obvious from the definition of $D(h)$ and the $\sim$ relation that whenever $h_1 \sim h_2$, also $D(h_1) = D(h_2)$. Consequently, $h_1 \overset{*}{\sim} h_2$ implies $D(h_1) = D(h_2)$.

For the other direction, assume that $D(h_1) = D(h_2)$. We shall transform h_2 to h_1 by a sequence of $\sim$ transformations as follows: Take the symbol in Σ_n *that is the first symbol in* h_1 *(i.e.,* $\pi^{-1}(1)$) and bring it to the first place of h_2 by successively switching it with all symbols preceding it in h_2; then take $\pi^{-1}(2)$ and bring it to the second position by switching it with all symbols preceding it, except $\pi^{-1}(1)$; and so on, until h_2 is transformed to h_1. It remains to show that all these switchings have been legal $\sim$ transformations. Suppose that at some time we had to switch σ_1 with σ_2 in a manner not allowed by Definition 3; that is either

a. $\sigma_1 = R_i$, $\sigma_2 = W_i$; this means, however, that in h_1, W_i precedes R_i, and hence h_1 is not a history.

b. $\sigma_1 = R_i$, $\sigma_2 = W_j$ and $S(R_i) \cap S(W_j) \neq \emptyset$. This would mean however, that (T_i, T_j) is in $D(h_2)$ and (T_j, T_i) is in $D(h_1)$. Since $D(h_1)$ and $D(h_2)$ have no cycles of length 2 we can conclude that $D(h_1) \neq D(h_2)$.

c. Similarly for $\sigma_1 = W_i$, $\sigma_2 = W_j$ and $S(W_i) \cap S(W_j) \neq \emptyset$. □

We can now prove the following Theorem.

THEOREM 2. A history $h = (n,\pi,V,S)$ is DSR if and only if $D(h)$ is acyclic.

Proof. Suppose that $D(h)$ is acyclic. We can thus sort topologically the set $\{T_1,\ldots,T_n\}$ of nodes of $D(h)$. Think of this order as a serial history h_S. It is immediate that $D(h_S) = D(h)$, and hence, by Lemma 4, $h \overset{*}{\sim} h_S$. It follows that h is DSR.

For the other direction, assume that h is DSR. We have two cases

a. $D(h)$ has a cycle (T_i, T_j, T_i) of length 2. This means that $\pi(R_i) < \pi(W_j) < \pi(W_i)$, and $S(R_i) \cap S(W_j) \neq \emptyset$, $S(W_i) \cap (S(W_j) \cup S(R_j)) \neq \emptyset$. It is easy to show that in all histories h' for which $h \overset{*}{\sim} h'$ we will also have $\pi'(R_i) < \pi'(W_j) < \pi'(W_i)$, as otherwise $h \not\equiv h'$, and $h \overset{*}{\not\sim} h'$, by Proposition 3. Hence there is no serial history h_S such that $h \overset{*}{\sim} h_S$, a contradiction.

b. $D(h)$ has no cycles of length 2. By Lemma 4, there is a serial history h_S such that $D(h) = D(h_S)$. However, serial histories h_S have acyclic $D(h_S)$, and hence $D(h)$ is acyclic. □

Theorem 2 suggests that histories that are DSR can be detected efficiently by checking $D(h)$ for acyclicity:

COROLLARY 1. Checking whether a history $h = (n, \pi, V, S)$ is DSR can be done in $O(|V|n^2)$ time. □

Also, we can rephrase Theorem 2 as follows (compare with Definition 4 below):

COROLLARY 2. A history $h = (n, \pi, V, S)$ is DSR if and only if we can find real numbers $\{s_1, \ldots, s_n\}$ such that

a. If $S(W_i) \cap S(R_j) \neq \emptyset$ and $\pi(W_i) < \pi(R_j)$ then $s_i < s_j$.

b. If $S(R_i) \cap S(W_j) \neq \emptyset$ and $\pi(R_i) < \pi(W_j)$ then $s_i < s_j$.

c. If $S(W_i) \cap S(W_j) \neq \emptyset$ and $\pi(W_i) < \pi(W_j)$ then $s_i < s_j$. □

4.2 The Class Q

DEFINITION 4. A history $h = (n,\pi,V,S)$ is in Q if there exist non-integer, distinct real numbers $S_1, S_2, \ldots, S_n$ with the following properties:

a. $\pi(R_i) < S_i < \pi(W_i)$

b. If $S(R_i) \cap S(W_j) \neq \emptyset$, $i \neq j$ and $\pi(R_i) < \pi(W_j)$ then $S_i < S_j$

c. If $S(W_i) \cap S(W_j) \neq \emptyset$ and $\pi(W_i) < \pi(W_j)$ then $S_i < S_j$. □

The real numbers $S_1, \ldots, S_n$ in Definition 3 are called *serializability points*. Their intuitive meaning is that the history h is the same as though transaction T_1 had executed indivisibly at the time instance S_1 (during which, by (a) above, it was active), transaction T_2 at S_2, and so on. As an illustration, the history

$$h = R_1[x]R_2[z]W_2[y]R_3[z]W_3[x]W_1[y]$$

is in the class Q, since the values $S_1 = 3.5$, $S_2 = 2.5$, and $S_3 = 4.5$ satisfy, as the reader can check, the requirements of the definition. The class Q was independently introduced by [Wo].

THEOREM 3. If h is in Q, then h is DSR.

Proof. Conditions (b) and (c) of the definition of the class Q above are identical to (b) and (c) of Corollary 2 to Theorem 2. Hence it suffices to show that condition (a) above implies condition (a) of Corollary 2. But this is immediate, because if $\pi(W_i) < \pi(R_j)$ we have that $S_i < \pi(W_i) < \pi(R_j) < S_j$, no matter what $S(R_j)$ and $S(W_i)$ are. □

Given a history $h = (n,\pi,V,S)$ we can construct another digraph $D'(h)$--a superdigraph of $D(h)$--with node set again $\{T_1,\ldots,T_n\}$ and (T_i,T_j) an arc if and only if one of the following holds

a. $\pi(W_i) < \pi(R_j)$

b. $\pi(R_i) < \pi(W_j)$ and $S(R_i) \cap S(W_j) \neq \emptyset$

c. $\pi(W_i) < \pi(W_j)$ and $S(W_i) \cap S(W_j) \neq \emptyset$.

In other words $D'(h)$ contains all the arcs of $D(h)$ and possibly some other arcs for the cases in which $\pi(W_i) < \pi(R_j)$ and $S(R_j) \cap S(W_i) = \emptyset$.

THEOREM 4. The history $h = (n,\pi,V,S)$ is in the class Q if and only if $D'(h)$ is acyclic.

Proof. Suppose that $h \in Q$, and let $s_1,\ldots,s_n$ be appropriate numbers. Without loss of generality $s_1 < s_2 < \cdots < s_n$. We shall show that whenever (T_i,T_j) is in $D'(h)$, then $i < j$. Suppose that $i > j$; by the definition of $D'(h)$ one of the following must hold:

a. $\pi(W_i) < \pi(R_j)$. However, $s_i < \pi(W_i) < \pi(R_j) < s_j$, which contradicts our assumption that $s_1 < s_2 < \cdots < s_n$ and $i > j$.

b. $\pi(W_i) < \pi(W_j)$ and $S(W_i) \cap S(W_j) \neq \emptyset$. By (c) of Definition 4, however, $s_i < s_j$, again a contradiction.

c. $\pi(R_i) < \pi(W_j)$ and $S(R_i) \cap S(W_j) \neq \emptyset$. Similarly, a contradiction is reached by (b) of Definition 4.

Consequently, $D'(h)$ is acyclic, since it is a subgraph of a total order.

For the other direction, suppose that $D'(h)$ is acyclic. We can sort topologically its nodes to obtain the order, say, $(T_1,T_2,\ldots,T_n)$. We can define the real numbers $s_1,s_2,\ldots,s_n$, and s_{n+1} (for convenience) as follows:

a. $S_{n+1} = 2n+1$

b. $S_j = \min\{S_{j+1}, \pi(W_j)\} - \frac{1}{n+1}$, $j = n, n-1, \ldots, 1$.

It is clear that the S_j's are distinct, increasing, non-integer real numbers, and that they satisfy (b) and (c) of Definition 4. It suffices thus to prove (a) of Definition 4, in particular that $S_i > \pi(R_i)$ for all i. Suppose that, for some i, $S_i \le \pi(R_i)$. Let j be the smallest index, no smaller than i, for which $\pi(W_j) < S_{j+1}$. Thus

$$S_i = \pi(W_j) - \frac{j-i+1}{n+1} > \pi(W_j) - 1$$

Consequently $\pi(R_i) > \pi(W_j) - 1$, or $\pi(R_i) > \pi(W_j)$. Hence $(T_j, T_i) \in A$, which contradicts the fact that $j \ge i$ in the topological sorting of $D'(h)$. □

COROLLARY. Testing whether a history $h = (n, \pi, V, S)$ is in Q can be done in $O(|V|n^2)$ time. □

4.3 Two-Phase Locking and the Protocol P3

A very influential proposal for guaranteeing serializability of update systems has been the two-phase locking mechanism of [EGLT]--also discussed extensively in [BS]. Also, the essence of a quite different serializability principle (which was used in the development of the SSD-1 distributed system [RG], [BGRP]) is captured by the so-called protocol P3 (see [BS]). In this Subsection we show that these two different philosophies of serializability are reduced, in our model, to two efficiently recognizable incommensurate subsets of our class DSR.

The two-phase locking strategy requests and releases actual locks--i.e., mechanisms that guarantee exclusive data access--during the execution of the different operations of an update. The rule that is proven sufficient for guaranteeing serializability is: never request a lock after a lock has been released. We have, therefore, two phases: one during which locks may only be requested, followed by one during which locks can only be released. The first release of a lock delimits the two phases. In our model of two-step updates the authors of [BS] note that two-phase locking for a history $h=(n,\pi,V,S)$ essentially amounts to dividing the interval from $\pi(R_j)$ to $\pi(W_j)$ into two intervals: one during which no symbol W_i with $S(R_j)\cap S(W_i)\neq\emptyset$ can exist, followed by one during which no symbol $\sigma\in\Sigma_n$ with $S(\sigma)\cap S(W_j)\neq\emptyset$ can exist. This is captured by the following definition:

DEFINITION 5. A history $h=(n,\pi,V,S)$ is *two-phase locked* (notation: $h\in 2PL$) if and only if there exist distinct non-integer real numbers $\ell_1,\dots,\ell_n$ (the *lockpoints*) such that

a. $\pi(R_i)<\ell_i<\pi(W_i)$ for $i=1,\dots,n$

b. If $S(R_i)\cap S(W_j)\neq\emptyset$, $i\neq j$ and $\pi(R_i)<\pi(W_j)$, then $\ell_i<\ell_j$

c. If $S(W_i)\cap S(W_j)\neq\emptyset$ and $\pi(W_i)<\pi(W_j)$, then $\pi(W_i)<\ell_j$. □

To understand Definition 5, consider a transaction (R_j,W_j) in a history $h\in 2PL$, and its lockpoint ℓ_j. The intuitive meaning of the lockpoint is the following: during the interval $[\pi(R_j),\ell_j]$ all variables in $S(R_j)$ are "protected" from writing by other transactions, by virtue of (b). Also during the interval $[\ell_j,\pi(W_j)]$ the variables in $S(W_j)$ are protected from reading *and* writing. Conditions (b) and (c) therefore essentially say that the interval $[\ell_j,\pi(W_j)]$ overlaps no interval $[\ell_k,\pi(W_k)]$ with $S(W_k)\cap S(W_j)\neq\emptyset$ and no interval $[\pi(R_k),\ell_k]$ with $S(W_j)\cap S(R_k)\neq\emptyset$. Thus, the second lock is granted before the first is released, in accordance with the two-phase locking principle.

Although Definitions 4 and 5 differ only slightly in condition (c), the latter is a substantial restriction. First, we notice that $2PL\subseteq Q$. Indeed, if $h\in 2PL$ then the lockpoints $\ell_1,\ldots,\ell_n$ are automatically valid serializability points $S_1,\ldots,S_n$ in Definition 4. To see this, just notice for that condition (c) of Definition 5 $(\pi(W_i)<\ell_j)$ together with (a) $(\ell_i<\pi(W_i))$ imply (c) of Definition 4 (namely, $S_i<S_j$). To show that the inclusion is proper, notice that for the history

$$h = R_1R_2R_3[x]W_1[x]W_2[y,z]W_3[y]$$

we have that $h\in Q$ (see Figure 5a for $D'(h)$) but $h\notin 2PL$. The explanation for the latter fact is that transaction 3 has no lockpoint ℓ_3, since, if it had, ℓ_3 should obey $\ell_3<\ell_1<4$ (by (b)) and also $\ell_3>5$ (by (c)).

We can, however, check very efficiently whether a history h is two-phase locked. Given any history $h=(n,\pi,V,S)$ we define the history $h^*=(2n,\pi^*,V,S^*)$, where h^* is obtained from h by inserting a transaction R_{n+j}, W_{n+j} after W_j in h for $j=1,\ldots,n$; $S^*(R_{n+j})=\emptyset$, and $S^*(W_{n+j})=S(W_j)$. For example, the history h^* for h of the example above is

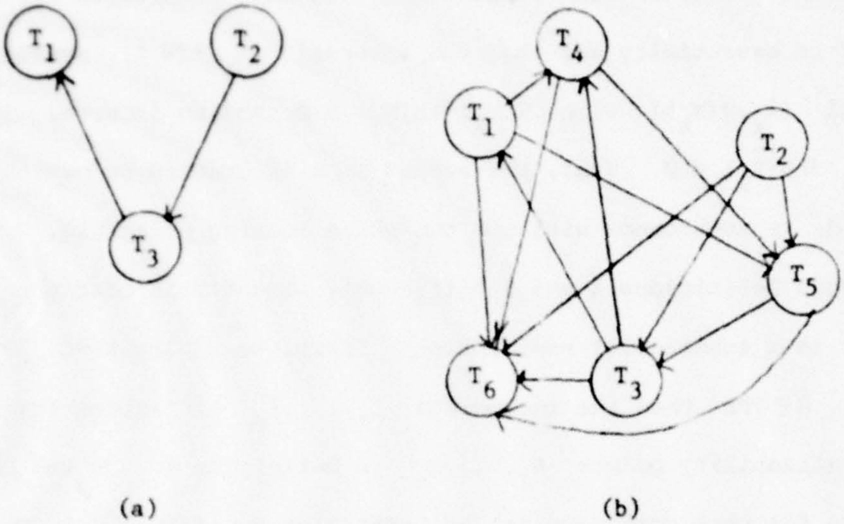


Figure 5

$$h^* = R_1R_2R_3[x]W_1[x]R_4W_4[x]W_2[y,z]R_5W_5[y,z]W_3[y]R_6W_6[y] \quad .$$

THEOREM 5. *For a history* $h = (n,\pi,V,S)$ $h \in 2PL$ *if and only if* $h^* \in Q$.

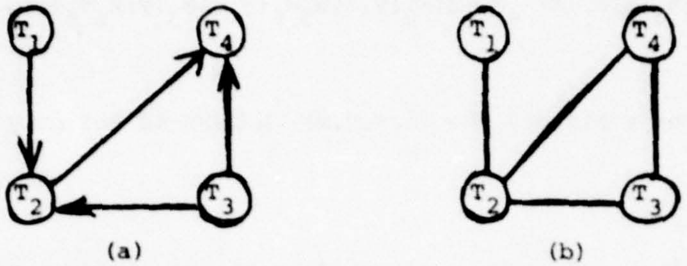
Proof. Let $\{\ell_1,\ldots,\ell_n\}$ be a set of distinct non-integer real numbers, and let $a(j)$ be the *number of positions to the right* that the symbol $\pi^{-1}(j)$ was shifted in h^*; in other words $a(j) = 2\cdot|\{W_i : \pi(W_i) < j\}|$. Consider the set $\{S_1,\ldots,S_{2n}\}$, where $S_i = \ell_i + a(\lceil \ell_i \rceil)$ for $i \leq n$, and $S_i = \pi(W_{i-n}) + a(\pi(W_{i-n})) + 3/2$ for $i > n$. We *claim* that $\{\ell_i\}$ is an acceptable set of lockpoints satisfying Definition 5 if and only if $\{S_j\}$ is a set of serializability points according to Definition 4. Both directions follow from the definitions. The formal derivation is omitted. □

To illustrate the theorem, the history h above is in Q, since $D'(h)$ is acyclic (Figure 5a). However, it is not in 2PL, because $D'(h^*)$ is *not acyclic* (Figure 5b). Naturally, Theorem 5 yields

COROLLARY. Testing whether a history $h = (n,\pi,V,S)$ is two-phase locked can be done in $O(n^2|V|)$ time. □

We now turn to formalizing and studying in our model the protocol P3 of [BGRP] and [BS]. Recall the digraph $D(h)$ defined for any history h in Subsection 4.1--see Figure 6a for an illustration in the case of

$$h = R_1[z]R_3W_3[x]R_2[x]W_1[z]R_4W_2[y,z]W_4[x] \quad .$$



(a) (b)

Figure 6

DEFINITION 6. Let $G(h)$ be the *undirected* graph corresponding to $D(h)$--Figure 6b. A *cycle* in $G(h)$ is a sequence (T_{i_1}, T_{i_m}) of $m \geq 2$ transactions such that $[T_{i_j}, T_{i_{j+1}}]$ are edges of $G(h)$, $j = 1,\ldots,m-1$, and so is $[T_{i_m}, T_{i_1}]$. Notice that all edges are cycles according to this definition. A cycle $(T_{i_1},\ldots,T_{i_m})$ is *bad* if

$$[S(R_{i_m}) \cup S(W_{i_m})] \cap S(W_{i_1}) \neq \emptyset,$$

and

$$S(R_{i_1}) \cap S(W_{i_2}) \neq \emptyset \quad .$$ □

Notice that in the above definition the first node of a cycle and the order of listing of the nodes are important. For example, in Figure 6 (T_1, T_2) is a bad cycle, whereas (T_2, T_1) is not. Bad cycles are, intuitively, those cycles that can correspond to a directed cycle in $D(h')$ for some other history h' involving the same transactions.

DEFINITION 6 (continued). Let $h = (n,\pi,V,S)$ be a history. We say that T_j is a *guardian* of T_i if there exists a bad cycle $(T_i, T_j, \ldots, T_k)$ in $G(h)$. We say that h *obeys the protocol* P3 (notation $h \in$ P3) if whenever T_j is a guardian of T_i we do not have $\pi(R_i) < \pi(W_j) < \pi(W_i)$.

□

For example, consider the history h of Figure 6. The only bad cycle in $G(h)$--Figure 6b--is (T_1,T_2), and hence the guardian relation is simple: just T_2 is a guardian of T_1. Since $\pi(W_2) > \pi(W_1)$, we have that $h \in P3$.

THEOREM 6. Suppose that $h = (n,\pi,V,S)$ is in P3. Then it is also in DSR.

Proof. We shall show that $h \in P3$ implies that $D(h)$ is acyclic. Suppose that $D(h)$ has a cycle $(T_1,T_2,\ldots,T_m)$, $m > 2$. Consider the arc (T_j,T_{j+1}) of $D(h)$--addition mod m; we have three cases:

a. $S(W_j) \cap S(W_{j+1}) \neq \emptyset$ and $\pi(W_j) < \pi(W_{j+1})$.

b. $S(W_j) \cap S(R_{j+1}) \neq \emptyset$ and $\pi(W_j) < \pi(R_{j+1})$.

c. $S(R_j) \cap S(W_{j+1}) \neq \emptyset$ and $\pi(R_j) < \pi(W_{j+1})$.

Notice that in both cases (a) and (b) we have that $\pi(W_j) < \pi(W_{j+1})$, and that more than one case may be applicable to the same arc. Case (c) is split into two subcases.

(c1) Cases (a) and (c) do not apply to the arc (T_{j-1},T_j).

(c2) $j = 1$, or case (a) or case (c) applies to (T_{j-1},T_j).

In case (c1) we have that $\pi(W_{j-1}) < \pi(R_j) < \pi(W_{j+1})$. In case (c2), however, we notice that T_{j+1} is a guardian of T_j. Consequently, since $\pi(R_j) < \pi(W_{j+1})$ we must necessarily have that $\pi(W_j) < \pi(W_{j+1})$.

Now, consider the operations O_j, $j = 1,\ldots,m$, where $O_j = R_j$ if case (c1) is applicable to the arc (T_j,T_{j+1}), and $O_j = W_j$ otherwise. We have shown that $\pi(O_j) < \pi(O_{j+1})$ for $j = 1,\ldots,m$ (addition mod m). This is a contradiction, since it implies that $\pi(W_1) < \pi(W_1)$. □

Theorem 6 implies the following, independently proved in [BS].

COROLLARY. Histories that obey the protocol P3 are serializable. □

Our next result concerns the complexity of recognizing those histories that obey protocol P3. By the definition of this class, this complexity is determined by the complexity of computing the guardian relation among the transactions in a history. We shall show how this relation can be computed efficiently. For each transaction T_j, let $\Gamma(T_j)$ be the set of all transactions T_i that satisfy $S(R_j) \cap S(W_i) \neq \emptyset$. Thus, $\Gamma(T_j)$ is the set of all transactions that are possibly guardians of T_j. To determine whether a transaction $T_i \in \Gamma(T_j)$ is indeed a guardian of T_j, we delete all edges $[T_j, T_k]$ such that $S(W_j) \cap [S(W_k) \cup S(R_k)] = \emptyset$ from $G(h)$, and then determine whether T_i and T_j are on the same biconnected component of the resulting graph. This can be done in $O(n^2)$ time by the algorithm of [Ta]. If T_i and T_j are on the same biconnected component, this means that there is a bad cycle $(T_j, T_i, \ldots, T_k)$ in $G(h)$, and hence T_i is a guardian of T_j; otherwise, it is not. Repeating this for all T_j's, we get an algorithm of total complexity $O(n^2(|V| + n^2))$. Hence we have

THEOREM 7. Testing whether a history $h = (n,\pi,V,S) \in P3$ can be done in $O(n^2(|V| + n^2))$ time. □

4.4 The Class SSR

Certain histories, though perfectly serializable, have a curious--and, according to some, undesirable--property. Consider, for example, the history

$$h = R_1[x]R_2W_2[x]R_3W_3[y,z]W_1[y] \quad .$$

This history is serializable. However, the only serial history equivalent to h is easily shown to be

$$h_S = R_3W_3[y,z]R_1[x]W_1[y]R_2W_2[x] \quad .$$

What is interesting is that in h transaction 2 has completed execution before transaction 3 has started executing, whereas the order in h_S has to be the reverse. This phenomenon is quite counterintuitive, and it has been opined that perhaps the notion of correctness in transaction systems has to be strengthened so as to exclude, besides histories that are not serializable, also histories that present this kind of behavior. This leads to the following definition:

DEFINITION 7. A history $h = (n,\pi,V,S)$ is said to be *serializable in the strict sense* (notation: $h \in SSR$). If there is a serial history $h_S = (n,\pi',V,S)$ such that $h \equiv h_S$, and $\pi(W_i) < \pi(R_j)$ implies $\pi'(W_i) < \pi'(R_j)$. □

It is not hard to verify that all histories in the class Q satisfy Definition 7. To see this, recall that a history h in Q has a set of serializability points $S_1 < S_2 < \ldots < S_n$, say, such that $h_S = R_1W_1 \cdots R_nW_n \equiv h$. Now, if $\pi(W_i) < \pi(R_j)$, we have, by the definition of S_i, $S_i < \pi(W_i) < \pi(R_j) < S_j$, and therefore $i < j$. Hence transactions i and j have the same order in h_S that they have in h. It follows that $Q \subseteq SSR$.

Nevertheless, the classes Q and SSR are not the same, as conjectured by [Wo]. A counterexample is

$$h = R_1[z]R_2[z]W_2[x,z]R_3[x]W_1[x,y]W_3[z]R_4[y]W_4[x] \quad .$$

This history is equivalent to the serial history

$$h_S = R_1[z]W_1[x,y]R_2[z]W_2[x,z]R_3[x]W_3[z]R_4[y]W_4[x] \quad ,$$

satisfying Definition 7. However, h is not in Q; to check this, just notice that the digraph $D'(h)$ shown in Figure 7 is not acyclic. It is not known whether the class SSR is efficiently recognizable.

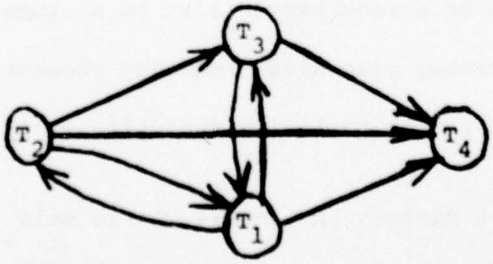


Figure 7

4.5 Summary

The topography of the set of all histories H and its subclasses SR, S (the serial histories), Q, SSR, DSR, P3 and 2PL is depicted in Figure 9. The inclusions shown either follow from the results of this section, or are straight-forward. We also show below an example of a history for each of the 12 regions in this diagram.

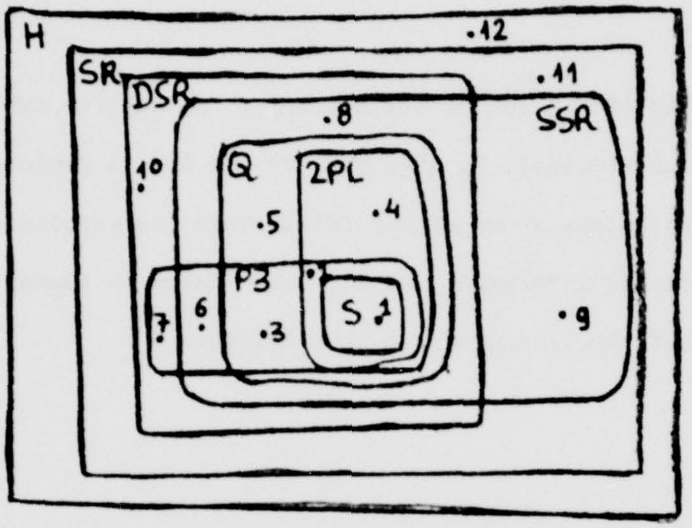


Figure 8

$h_1 \;=\; R_1[x]W_1[x]R_2[x]W_2[x]$

$h_2 \;=\; R_1[x]R_2[y]W_1[x]W_2[y]$

$h_3 \;=\; R_1R_2R_3[x]W_1[x]W_2[y,z]W_3[y]$

$h_4 \;=\; R_1[x]R_2W_2[x,y]W_1[z]R_3W_3[y,z]$

$h_5 \;=\; h_3 \circ h_4$

$h_6 \;=\; R_2[z]R_1W_2[x,z]R_3[x]W_3[z]W_1[x,y]R_4[y]W_4[x]$

$h_7 \;=\; R_3[x]R_1W_1[x]R_2[y]W_2W_3[y]$

$h_8 \;=\; R_2[z]R_1[z]W_2[x,z]R_3[x]W_1[x,y]W_3[z]R_4[y]W_4[x]$

$h_9 \;=\; R_1R_3W_3[x]R_2[x]W_1[x]W_2[x]$

$h_{10} = h_7 \circ h_4$

$h_{11} = h_7 \circ h_9$

$h_{12} = R_1[x]R_2[x]W_1[x]W_2[x]$

5. RESTRICTIONS ON THE READ- AND WRITE-SETS

It turns out that if we impose certain restrictions on the structure of the map S of a history--i.e., the read- and write-sets of the transactions in the history--the topography of H (shown in Figure 8 for the general case) is simplified considerably. The most striking such result is that of [SLR]. A basic assumption in the model of [SLR]--which is otherwise more general than the present in that it allows more than two steps--is that no database entity (or variable) is updated, unless it has been previously read. In our model and notation, this means that $S(W_j) \subseteq S(R_j)$. What is surprising, is that serializability, an NP-complete predicate in our model, is efficiently decidable in theirs. We explain this in view of our previous discussion as follows:

THEOREM 7. Suppose that for a history $h = (n,\pi,V,S)$ we have $S(W_j) \subseteq S(R_j)$ for $j = 1,\ldots,n$. Then h is serializable if and only if h is in DSR.

Proof. It suffices to show that if $S(\sigma_1) \cap S(\sigma_2) \neq \emptyset$ and $\pi(\sigma_1) < \pi(\sigma_2)$ for $\sigma_1, \sigma_2 \in \Sigma_n$ such that at least one of σ_1, σ_2 is a write symbol, then $\pi'(\sigma_1) < \pi'(\sigma_2)$ in any history (n,π',V,S) equivalent to h. Suppose that $\sigma_1 = W_1$, $\sigma_2 = W_2$. $S(W_1)$ and $S(W_2)$ share a variable x, which, by hypothesis, is also in $S(R_1)$ and $S(R_2)$. Consequently, in h T_2 reads x *from either* T_1 or from another transaction which, by the same argument, reads x from another, and so on, up to T_1. Now, notice that the $S(R_j) \supseteq S(W_j)$ assumption implies that in any serializable history there can be no dead transactions. Hence, by Proposition 1, in any history (n,π',V,S) equivalent to h we must also have $\pi'(W_1) < \pi'(W_2)$. The other two cases are settled very similarly. □

It turns out that the rest of the classes of histories discussed previously have a considerably simpler structure under the assumption that $S(W_j) \subseteq S(R_j)$. We show below, without proofs the corresponding diagram.

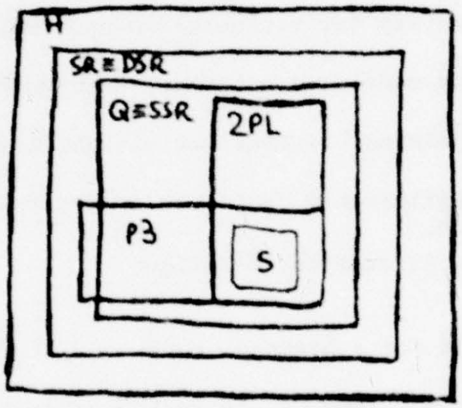


Figure 9

Under a different restriction on S, the class SSR coincides with SR.

THEOREM 8. Suppose that in a history $h = (n,\pi,V,S)$ there is a subset $X = \{x_1,x_2,\ldots,x_n\} \subseteq V$ such that for $j = 1,2,\ldots,n$ we have (a) $X \subseteq S(R_j)$, (b) $x_j \in S(W_i)$ if and only if $i = j$. Then h is serializable if and only if $h \in SSR$.

Sketch of Proof. Imagine that the variable x_j is a Boolean signalling whether transaction T_j has completed. Therefore, if T_j completed in h before T_i started, the same must hold in any other history equivalent to h. □

6. SCHEDULERS OF HISTORIES

The practical importance of the classes of histories 2PL and P3 discussed in Section 4 stems from the fact that they are known to correspond to simple *schedulers*. A scheduler for a class of histories (to be defined formally below) is generally an algorithm that takes as an input an arbitrary history--possibly non-serializable--and returns a history which is the "closest" to the given one among those belonging to the class. If the class is a subset of SR, therefore, the scheduler guarantees that its output history is serializable. Such a scheduler can be used in the serializability component of the database management system. Of course, in practice one would expect that a scheduler operates on-line and is reasonably efficient.

The history-input of the scheduler is the sequence of arriving user requests. The output of the scheduler is the actual execution sequence. The basic fact that makes our approach very different from previous work on concurrency control which was motivated by operating systems (e.g., the notion of *determinacy* of [CD]) is that the supplier of this input history is a population of users, each user being unaware of the actions of the others. This implies that the order of arrival of these requests has no semantic content whatsoever, and therefore the scheduler is not bound to produce an output which is equivalent (or related in any prescribed way) to the input. In fact, the operation of the scheduler becomes interesting and important exactly when the scheduler must necessarily transform the input to an inequivalent output, because the input is non-serializable, say.

There are, however, certain performance criteria that the input-output mapping of a scheduler should satisfy. For example, a trivial scheduler which guarantees serializability is the one that outputs only serial histories. This is, however, too restrictive a mechanism to be of practical value. Intuitively, the richer the output class, the more powerful the scheduler, because a less restrictive class of histories will require less reshuffling of the operations and will cause fewer and shorter unnecessary delays. Ideally, we would like to have a *serializer*, whose output spans all of SR. Unfortunately, we shall soon see that the existence of such a practically useful device is very improbable.

DEFINITION 8. The metric $d(.,.)$ on the set H is defined as follows:

a. $d((n,\pi,V,S),\ (n,\rho,V,S)) = n\text{-}\max\{j:\pi^{-1}(i) = \rho^{-1}(i),\ i=1,\ldots,j\}$.

b. $d((m,\pi,V,S),\ (n,\rho,W,T)) = \infty$ if any one of $m \neq n$, $V \neq W$, $S \neq T$ holds. □

The distance between two histories defined on the same set of transactions is therefore n minus the length of their longest common prefix. Notice that $d(.,.)$ satisfies the metric axioms. A variety of other metrics would suffice for what follows.

DEFINITION 8 (continued). Let C be a non-empty subset of H. A *scheduler for* C is a function $A_C : H \to C$ such that

$$d(h, A_C(h)) = \min\{d(h,h') : h' \in C\} \quad .$$ □

Thus, A_c can be thought of as *projecting* H onto C under the metric d(.,.). Notice that $A_c(h)$ and h will not be equivalent in general. The metric d(.,.) requires that A_c leaves histories in C intact, and, in fact, it leaves intact as long prefixes of arbitrary histories as possible.

Let us restate now the assumptions of our model of schedulers

(a). A scheduler A_c minimizes the d-distance between its input and its output. This intuitively means that the scheduler operates on-line, and, furthermore, that it acts in an *optimistic* way: As long as the history seen so far could possibly be extended to a correct history (here by "correct history" we mean one which the scheduler, in its limited sophistication, recognizes as correct, or, equivalently, an element of $C = A_c(H)$) the scheduler does not intervene to rearrange read and write requests. *As a corollary, if the scheduler is fed with its own output, it leaves it intact; it is therefore idempotent*, or a projection.

This is a quite reasonable assumption to make. Although we cannot totally exclude the possibility of schedulers that operate otherwise (for example, anticipating future requests that will make the history non-serializable), all schedulers proposed in the past satisfy this assumption. Any scheduler implemented by natural constructs such as locks [KP], [EGLT] or queues has this property.

(b). Among all histories in C that have the longest possible common prefix with the input history, A_c selects any one as its output. Clearly, in practice this choice would be made so as to minimize some more refined metric d'. However, the results obtained below for our weaker metric d' would apply to more relaxed metrics, too.

We say that A_c is an *efficient scheduler* if A_c is computable in polynomial time. Our goal in this Section is to understand which classes of histories have efficient schedulers. It is tempting to conjecture that if a class is in P, then it has an efficient scheduler. This conjecture is not plausible, because, consider the following:

EXAMPLE. Let $E = \{h \circ h_S : h_S$ is serial, and $h \equiv h_S\}$.

Obviously, E can be recognized in polynomial time; the algorithm involves splitting a given history in two halves, testing whether the second half is serial, and whether the second half is equivalent to the first. However, it is also easy to see that E *cannot* have an efficient scheduler, unless $P = NP$. Suppose that E has an efficient scheduler A_E. Then we could test whether an arbitrary history h is serializable by first computing $A_E(h \circ h)$, and then checking whether $A_E(h \circ h)$ starts with h. Since A_E is supposed to leave unchanged as long prefixes of its input as possible, it will alter the first half of $h \circ h$ only if h is not serializable. Since serializability is known to be NP-complete, E cannot have an efficient scheduler unless $P = NP$. □

Our next result essentially says that efficiently recognizable classes have efficient schedulers, unless they are as pathological as our example E above. Let $h = (n, \pi, V, S)$ be a history, considered now as a string of symbols representing n, V, S and the permutation π. A *prefix of* h *is an initial segment of this representation*, containing the encoding of n, V, S, as well as an initial part of π--i.e., $<\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(j)>$ for some $0 \leq j \leq 2n$. If C is a class of histories, then $PR(C)$ is the set of all prefixes of all histories in C.

THEOREM 9. Let C be a subset of H. C has an efficient scheduler if and only if $PR(C) \in P$.

Proof. Suppose that C has an efficient scheduler A_C. In order to determine whether a string g is a prefix of a history $h \in C$ we may act as follows: we first verify that g contains encodings of n, V, and S, together with an initial segment ρ of a permutation π of Σ_n. We then generate a *completion* $\bar{\rho}$ of ρ by juxtaposing to ρ the symbols W_j such that R_j but not W_j is present in ρ, and then the strings R_jW_j for all j's such that neither R_j nor W_j appears in ρ. We then calculate $h' = A_C((n,\bar{p},V,S))$. It is straightforward to see that g is a prefix of h' if and only if $g \in PR(C)$. Thus we can efficiently determine whether $g \in PR(C)$.

For the other direction, suppose that $PR(C) \in \mathcal{P}$. Based on the recognition algorithm for $PR(C)$ we design an efficient scheduler: A_C shown in Figure 10. A_C computes $A_C(h) = (n,\rho,V,S)$ by determining ρ element-by-element. It should be obvious that A_C operates as prescribed within a time bound of $O(n^2C(n,|V|))$, where $C(n,|V|)$ is the complexity of recognizing $PR(C)$. The Theorem follows. □

It is now easy to link the discussion of Sections 3 and 4 with the existence of efficient schedulers. We get two types of results:

COROLLARY 1. Unless $\mathcal{P} = \mathcal{NP}$, SR has no efficient scheduler. □

COROLLARY 2. The classes S, 2PL, P3, Q, DSR have efficient schedulers.

Proof. We have shown that these sets are in $\mathcal{P}$; it is usually straightforward to show that their sets of prefixes are also in $\mathcal{P}$ (this is not a general property of $\mathcal{P}$; there are languages in $\mathcal{P}$ that have non-recursive sets of prefixes). As an illustration, we will sketch a proof that $PR(P3) \in \mathcal{P}$. First, given an encoding of n, V, S, and a segment ρ of π, we first compute from S the digraph F of the guardian relation among $\{T_1,\ldots,T_n\}$. We next make sure that whenever T_j is a guardian of T_i and $\rho(W_j)$ is defined, then either $\rho(W_i) < \rho(W_j)$, or $\rho(R_i) > \rho(W_j)$, or $\rho(R_i)$ is undefined. Finally, we make sure that ρ

Scheduler A_C

Input: a history $h = (n,\pi,V,S)$

Output: a history $h' = (n,\rho,V,S) \in C$ such that $d(h,h')$ is the smallest possible, if such an h' exists.

```
begin
if  $(n,< >,V,S) \notin PR(C)$  then return
   comment < > is the empty permutation;
else begin
     $\rho := < >$;
     for  $j = 1,\ldots,2n$  do
        begin
        done: = false;
        for  $i = j, j+1,\ldots,2n$  do until done
             if  $(n,<\rho,\pi^{-1}(i)>,V,S) \in PR(C)$  then
                begin
                done: = true;
                interchange $\pi^{-1}(i)$ and $\pi^{-1}(j)$;
                $\rho: = <\rho,\pi^{-1}(i)>$;
                end;
        end;
     end;
return  $(n,\rho,V,S)$;
end
```

Figure 10

can be completed in a manner not violating P3. It turns out that this amounts to verifying that the restriction of F to the transactions that are active (i.e., $\rho(R_j)$ is defined but $\rho(W_j)$ is not) is acyclic (a discussion of this part follows the proof). Hence we have an efficient algorithm for PR(P3). □

We show in Figure 11, without proofs, stylized versions of efficient schedulers for the classes 2PL (11b). P3 (11a), DSR and Q (11c; for Q we also include the two statements labeled Q). Besides serializability, these algorithms must also guarantee the absence of *deadlocks*. The issue of deadlocks appears to be orthogonal to that of serializability, and, in fact, clever serializability methods are known to introduce increased danger of deadlocks of the "circular waiting" variety ([CD], pp.40-60). A unified treatment of serializability and deadlocks in a restricted data model is attempted in [SK]. In all cases of interest to us, deadlocks can be prevented by testing a dynamically changing *deadlock graph* for acyclicity. For example, in two-phase locking deadlock can occur if a number of transactions have each locked their read-set, and are awaiting for each other to release their locks. Hence, in this case the deadlock graph has variables as nodes, and has an arc from x to y if and only if some transaction currently on phase 1 reads x and writes y. In P3 the deadlock graph is the restriction of the guardian relation to the currently active transactions--this was mentioned in the proof of Corollary 2 to Theorem 9. Finally the deadlock graph in DSR (resp., Q) has as nodes the active transactions and includes the arc (T_i,T_j) if and only if there is a path from T_i to T_j in D(h)--resp. D'(h)--and $S(W_i) \cap S(W_j) \neq \emptyset$.

Our notation in Figure 11 assumes that the process R_j or W_j is initiated as soon as a corresponding read or write requests arrive. We use constructs such as *when* (denoting the awaiting for a condition) and *ibegin...iend* (bracketing statements that are to be executed indivisibly). It should be obvious that these algorithms can be implemented deterministically and efficiently on any standard model of computation.

```
process $R_j$
when the deadlock graph with $T_j$ is acyclic do
    output ($R_j$)

process $W_j$
when $T_j$ is not the guardian of an active transaction do
    output ($W_j$)
```

(a)

```
process $R_j$
when the deadlock graph with $T_j$ is acyclic and
        no variable is $S(R_j)$ is read-locked do
ibegin
    write-lock all variables in $S(R_j)$;
    output ($R_j$)
iend;
when a process $W_i$ with $S(W_i) \cap S(R_j) \neq \emptyset$ or $i = j$ has been initiated and
      no variable in $S(W_j) - S(R_j)$ is writelocked do
ibegin
    write-lock and read-lock all variables in $S(W_j)$;
    un-write-lock all variables in $S(R_j) - S(W_j)$.
iend

process $W_j$
when $R_j$ has terminated do
    ibegin output ($W_j$)
          unlock all variables in $S(W_j)$
    iend
```

(b)

```
process $R_j$
declare $L_j$ sequence of symbols in $\Sigma_n \cup \{f\}$
comment $L_j$ contains all $R_i$ or $W_i$ such that $T_i$ is reachable by a
      path from $T_j$ in D (resp. D'), up to this point;
when the deadlock graph is acyclic and for no $T_i \neq T_k$
     with $S(R_j) \cap S(W_i) \neq \emptyset$, $S(R_j) \cap S(W_k) \neq \emptyset$ is $W_i \in L_k$ do
ibegin
     output $(R_j)$
     $L_j := \{R_j\}$
     add $R_j$ to all $L_k$ containing $W_i$ with $S(R_j) \cap S(W_i) \neq \emptyset$
  Q: add $R_j$ to all $L_k$ containing f
iend

process $W_j$
when the deadlock graph contains no arc $(T_i, T_j)$ do
     ibegin
     output $(W_j)$
     add $W_j$ to all $L_k$ containing $\sigma$ such that $S(W_j) \cap S(\sigma) \neq \emptyset$
  Q: add f to all $L_k$ containing $R_j$ or $W_j$
     set $L_j := \emptyset$
     iend
```

(c)

Figure 11

7. DISCUSSION

We shall consider extensions of our results in three directions: general multi-step transactions, interpreted transactions, and distributed databases.

7.1 Multi-step Transactions

We shall briefly discuss how our entire development of Sections 2 through 6 can be easily extended to a far more general multi-step model of transactions. We consider transactions that consist of sequences of steps; each step may involve both reading and writing. The values written must be considered as *uninterpreted functions* of all variables read at the present or previous steps of the same transaction. Our definition of liveness now applies to individual *steps* of transactions. No further modifications are necessary for stating the analog of Proposition 1.

Serializability is obviously NP-complete in this model, as it subsumes ours. Assuming that no transaction reads intermediate results of another or reads two different versions of the same variable at two different steps--in which case the history is not serializable--Lemma 2 is also valid. The four serializability principles discussed in Section 4 remain virtually unchanged--in fact, two-phase locking was initially proposed for a similar model in [EGLT]. For another example, we shall describe in a somewhat more detailed manner the generalized P3 class of histories. In the multi-step model a step s of a transaction can be an *(i,j)-guardian* of another transaction, where $i < j$ are steps. This means that s *interacts* with i--i.e., either its write set includes variables of i, or vice-versa--and there is a chain of interactions from s to j. If this is the case, s is not allowed to occur between i and j. This P3 protocol always yields DSR (and hence serializable) histories. For the classes DSR and Q, we have similar graphs $D(h)$ and $D'(h)$. An arc (T_i,T_j) is in $D(h)$ if a step of T_i interacts with a subsequent step of T_j. For $D'(h)$, it may just be that the last step of T_i precedes the first step of T_j. The acyclicity of $D(h)$ again guarantees serializability, and that of $D'(h)$ strict serializability. Hence, these remain two most general serializability techniques, subsuming two-phase locking and P3, in this general setting, too.

Finally, it is easy to see that the results of Section 6--the necessary and sufficient condition for the existence of efficient schedulers and its corollaries--apply even more directly to multi-step histories. We hope that the reader is by now convinced that introducing general multi-step transactions would have resulted in an unmanageably cumbersome notation but in very few new important ideas.

7.2 Interpreted Transactions

A significant departure from our model would be to look more closely into the computations performed by the transactions and exploit their details for studying serializability--or correctness, in general. If only syntactic information about the transactions is available (e.g., the read- and write-sets) then serializability can be formally proved to be the right concurrency concept [KP]. If, however, semantics of the functions performed, or even the integrity constraints, are known, then it may be the case that more liberal concurrency principles than serializability are applicable. An example is the correctness theory proposed in [Lal], where the concurrency control mechanism takes into account information about the semantics and integrity constraints supplied by correctness proofs of the individual transactions. The extent to which such information is helpful is investigated in [KP].

It is doubtful whether complete semantic information can be used effectively for concurrency control. Any reasonably complex domain of interpretation (e.g., arithmetic) would soon make the serializability problem undecidable. There should be, however, ways to use partial semantic information in order to improve our understanding of serializability. One possibility is to use the fact that two transactions perform precisely the same function; one of the implications is that they commute. It is not too hard to see that this adds nothing to the model developed thus far. Incidentally, this allows us to extend our original model so as to permit multiple occurrences of a transaction in a history.

Another possibility would be to selectively consider certain very simple transactions to be interpreted. A good example of a very common transaction that performs a well-understood function is the

a transaction that reads x and later records its value at y. Serializability become trickier. For example the history

$$h = R_1[x]R_2R_3[x]W_2[x]W_3[y]R_4[y]W_4[x]R_5[x]W_5[z]W_1[z]$$

is not serializable in our ordinary sense, but becomes equivalent to the serial history $h_S = T_5T_1T_2T_3T_4$ once we assume that transactions 3 and 4 are copiers. Proposition 1 becomes somewhat more complex in the presence of copiers. However, it is interesting to note that if copiers are restricted not to read variables from other copiers, then the introduction of copiers adds no strength to our model, and Proposition 1 and Lemma 2 remain unchanged under this assumption. This remark plays an important role in the next topic of our discussion.

7.3 Distributed Databases

There is a large body of literature aiming at the understanding of the quite elusive notion of distributed computing (see, for example [La2]). Distributed databases have inherited some of the intricacies of this area [RG], [Th]. We shall limit our discussion to the case of two complete copies of the database in different locations, although there are difficulties which first appear in the cases of three copies or of selective redundancy [BSRG]. A major problem is, what happens when a transaction is run in one location, thus changing only one of the two copies. A simple technique for solving this would be to send an *update message* [BGRP] to the other location as soon as the transaction has completed. We have therefore a sequence of genuine transactions and update messages running in the system, and we can thus view the two copies of the database as a single database--think of the two copies of the variable x as two variables x_1 and x_2.

A difficulty appears when we try to define a history. The distributed nature of our computation, the communication delays and imperfect clocks make temporal priority--on which our ordinary notion of history was based--less tangible. The observation here is that mistakes in our arrangement of the events which are due to the above factors preserve history equivalence. Hence, we can put together a history--the *global log* of [BGRP]--as long as it is consistent with local priorities and arrivals of messages. Now, the update messages are in fact just copiers, and they only read variables that were updated by ordinary transactions. Hence the last remark of the previous Subsection is applicable, and the serializability problem has been reduced to the one already studied! Of course, we are not just looking for serializability, but for the existence of an equivalent serial history in which an update message immediately follows the corresponding transaction. This, however, does not change the essence of the task. All our special case results hold with very minor modifications.

What is considerably more complex in the distributed context is the subject of schedulers. There is no obvious neat way to compile syntactic restrictions on the global history into distributed algorithms that achieve them. It therefore appears that distributed history schedulers must concern themselves with the details of the underlying model of distributed computation in order to implement the intended serializability principle; the formidable algorithms of [Th] and [BSRG] illustrate this point. Nevertheless, it is still natural to conjecture that the more general ideas related to the classes DSR and Q would prove advantageous in the distributed environment as well.

7.4 Open Problems

We have proposed a formalism for the concurrency control problem for databases. There are two aspects of this formalism that may limit its applicability, and must therefore be modified in a second attempt. One is our basic assumption, manifested throughout the paper, that the syntactic description of all transactions to occur in the history is known to the scheduler a priori. It is not clear how to remove this assumption, and still retain the wealth of available solutions. One way would be to have, following [BSRG], a certain number of prototype transactions--or *classes* -- to one of which any arriving transaction can be matched. Another way out would be to adopt only *transaction-driven* concurrency controls. Two-phase locking [EGLT] is an example of such a concurrency control, and so would be any other locking scheme. The limitations of such approaches are studied in [KP]. On the other hand, it is possible that variants of the schedulers presented here could also be implemented in a transaction-driven manner.

Secondly, our way of evaluating the performance of schedulers is also in need of an improvement. We propose only a qualitative measure of the performance of a scheduler--namely the set of all output histories. This leads to only a partial order of schedulers. This was shown to be a reasonable and useful approximation of reality when the goal is to derive indicative results or compare general principles of serializability. It is clear, however, that a more concrete measure of performance is needed for more practical applications. One promising direction would be to somehow count the total number of delays imposed on requests--at a first approximation, the number of transaction steps that cannot execute immediately upon arrival.

This would be a refinement of our measure: our measure, roughly speaking, assigns a perfect score to all histories that remain the same, and zero score to all histories that are changed, however small the change. A more refined measure might even put to test some of our assumptions, like the "optimistic scheduler" assumption (Section 6): in certain cases it may be preferable to intervene and modify slightly the history, when serializable completion becomes extremely unlikely, although not impossible. Naturally, adopting a more concrete measure of performance for schedulers will most likely require the introduction of specific and pragmatic details of the particular application, and the overall approach may have to be probabilistic.

By considering only serializability as our notion of correctness we have somehow limited our scope. Examples of concurrency control techniques more general than serializability can be found in [Lal] and [KL]. They are arrived at by assuming that the scheduler has more than syntactic information about the transaction system that it handles--e.g., semantic information or understanding of the integrity constraints. It is pointed out in [KP] that serializability is just one point in the trade-off between information and performance of schedulers. However, we feel that there is something natural about the use of syntactic information for concurrency control, and the importance of concurrency techniques stronger than serializability is of limited practical value.

Finally, we recall two other problems that are left open here: the complexity of recognizing the class SSR, and developing techniques for designing distributed schedulers from syntactic specifications.

ACKNOWLEDGMENT

Many illuminating discussions with Phil Bernstein have influenced this work. Also, we acknowledge helpful discussions with H.T. Kung, Dan Rosenkrantz, Jim Rothnie, and Jeff Ullman, and careful reading of the manuscript by Marco Casanova and an anonymous referee.

REFERENCES

[AHU] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison, 1975.

[BGRP] P. A. Bernstein, N. Goodman, J. B. Rothnie, C. H. Papadimitriou. "Analysis of Serializability of SDD-1: A System of Distributed Databases (the Fully Redundant Case)", *IEEE Trans. on Software Engrg.*, Vol. SE-4, 3, May 1978.

[BPR] P. A. Bernstein, C. H. Papadimitriou, J. B. Rothnie. "Resolving Certain Concurrent Update Problems Without Locking: An Abstract", *Proc. IEEE Workshop on OS and DBMS*, Chicago, 1977.

[BS] P. A. Bernstein, D. W. Shipman. "A Formal Model of Concurrency Control Mechanisms for Database Systems", *Proc. 1978 Berkeley Workshop on Distributed Databases and Computer Networks*, September 1978, Berkeley.

[BSRG] P. A. Bernstein, D. W. Shipman, J. B. Rothnie, N. Goodman. "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the General Case)". TR CCA-77-09, Computer Corp. of America, Cambridge.

[CD] E. G. Coffman, P. Denning. *Operating Systems Theory*, Prentice Hall, 1973.

[EGLT] K. P. Eswaran, J. N. Gray, R. A. Lorie, I. L. Traiger. "The Notions of Consistency and Predicate Locks in a Database System", *CACM*, 19, 11, 1976.

[GJ] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[Ka] R. M. Karp. "Reducibilities Among Combinatorial Problems", in *Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher (eds.), Plenum, 1972.

[KL] H. T. Kung, P. L. Lehman. "A Concurrent Database Problem: Binary Search Trees", *Proc. 4th Intern. Conference on Very Large Databases*, Berlin, 1978.

[KP] H. T. Kung, C. H. Papadimitriou. "An Optimality Theory of Concurrency Control for Databases", Manuscript, Carnegie-Mellon University, 1979.

[La1] L. Lamport. "Towards a Theory of Correctness for Multi-User Data Base Systems", TR CA-7610-0712, Mass. Computer Associates, 1976.

[La2] L. Lamport. "Time, Clocks and Ordering of Events in a Distributed System", Comp. Assoc. TR CA-7603-2911, 1976.

[LPP] D. C. Luckham, D. M. R. Park, M. S. Paterson. "On Formalized Computer Programs", *JCSS*, 4, 3, pp. 220-249, 1970.

[PBR] C. H. Papadimitriou, P. A. Bernstein, J. B. Rothnie. "Computational Problems Related to Database Concurrency Control", *Proc. Conf. on Theor. Comp. Sci.*, Univ. of Waterloo, 1977.

[PS] C. H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization Algorithms*, in preparation.

[RG] J. B. Rothnie, N. Goodman. "An Overview of the Preliminary Design of SDD-1: A System of Distributed Databases", *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, California, May 1977.

[SK] A. Silberschatz, Z. Kedem. "Consistency in Hierarchical Database Systems", Manuscript, U. of Texas at Dallas, 1978.

[SLR] R. C. Stearns, P. M. Lewis, D. J. Rosenkrantz. "Concurrency Control for Database Systems", *Proc. 16th FOCS*, 1976, pp. 19-32.

[Ta] R. E. Tarjan. "Depth-first Search and Linear Graph Algorithms", *Siam J. Computing*, 1, 2, pp. 146-160, 1973.

[Th] R. H. Thomas. "A Solution to the Update Problem for Multiple Copy Databases which Uses Distributed Control", TR 3340, BBN, Cambridge, 1976.

[Wo] W. Wong. "Analysis of Serializable Logs", Manuscript, Harvard University, 1978.